ELSEVIER

Information Sciences 143 (2002) 147–158

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

Two-dimensional packing algorithms for layout of disconnected graphs

Ugur Dogrusoz *,[1]

Computer Engineering Department, Bilkent University, Ankara 06533, Turkey

Abstract

We present and contrast several efficient two-dimensional packing algorithms for specified aspect ratio. These near-linear algorithms are based on strip packing, tiling, and alternate-bisection methodologies and can be used in the layout of disconnected objects in graph visualization. The parameters that affect the performance of these algorithms as well as the circumstances under which they perform well are analyzed. © 2002 Elsevier Science Inc. All rights reserved.

Keywords: Graph visualization; Graph layout; Two-dimensional packing

1. Introduction

Graph drawings model and help us visualize the complex information in a system of discrete objects and their relationship. *Graph layout* is the automatic positioning of the nodes and edges of a graph in order to produce an aesthetically pleasing drawing that is easy to comprehend [1].

Many graph layout and editing systems have been developed in the past [1]. One essential aspect that has not been addressed sufficiently in any previous system is the layout of disconnected graphs; that is, the placement of the isolated nodes and components of a disconnected graph. Disconnected graphs

* Tel.: +90-312-290-1612; fax: +90-312-266-4126.
E-mail address: ugur@cs.bilkent.edu.tr (U. Dogrusoz).

[1] Research supported in part by NIST, ATP grant number 70NANB5H1162 and Tom Sawyer Software, 804 Hearst Avenue, Berkeley, CA 94710, USA.

PII: S0020-0255(02)00183-4

occur rather frequently in real life applications either during the construction of a graph interactively or because of the nature of the application (Fig. 1).

Most graph layout algorithms assume a graph to be connected and try to minimize the area needed for the resulting drawing (i.e., the area of the smallest rectangle bounding the drawing). No matter how effective an algorithm is, in minimizing the area needed for the drawing of a connected graph, the space wasted overall could be arbitrarily large if the relative locations of the isolated nodes and components of a disconnected graph are chosen by a naive, inefficient method. Another key parameter here is the aspect ratio of the region (e.g., a window) within which the graph is to be displayed (Fig. 2). When displaying a graph, the more the wasted space is, the less visible objects will be, making the visualization process more difficult. Thus, a disconnected graph layout algorithm must strive for a packing of disconnected objects which respects the aspect ratio of the region in which it is to be displayed as well as for a tight packing of these objects.

In this paper, we present efficient two-dimensional packing algorithms for the layout of disconnected graphs for a specified aspect ratio based on *strip packing*, *tiling*, and *alternate-bisection* methodologies. We analyze these algorithms both theoretically and experimentally. Furthermore, we contrast their performance with respect to each other as well as analyzing how their performance depends on certain parameters.

Fig. 1. An example of a disconnected graph.

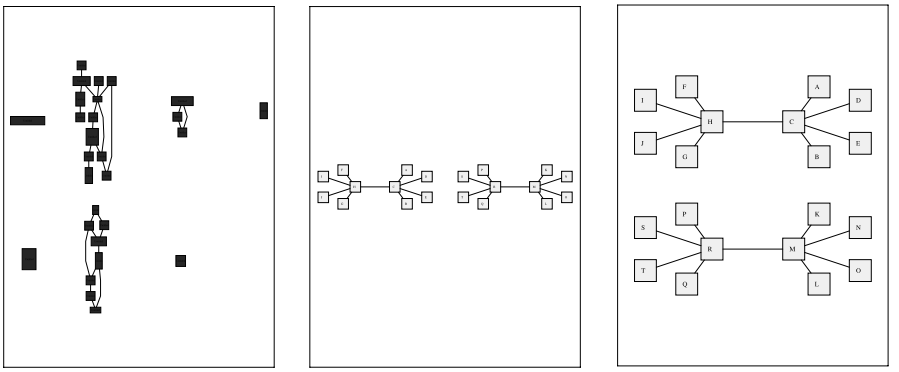

Fig. 2. How a naive disconnected graph layout algorithm makes inefficient use of the area (left) and why the aspect ratio of the region in which the graph is to be drawn should be taken into account during disconnected graph layout (right).

2. Definitions and basics

Throughout the paper, for simplicity we assume individual nodes and graph components are represented with rectangles in terms of the area they require in a drawing; and, the terms "rectangle" and "graph object", or simply "object", are used interchangeably. The tightest rectangle bounding a graph's drawing is said to be the *bounding rectangle* of the drawing. The size of a graph's drawing is identified with its bounding rectangle's. The *aspect ratio* of a rectangle $R = (W, H)$ is equal to W/H.

A disconnected layout algorithm's task then is to position a set of rectangles with ordered dimensions such that no pair of rectangles overlap and the area of the bounding rectangle of the drawing is minimized, respecting the aspect ratio of the region in which the graph is to be displayed. There has been extensive research done on two-dimensional packing, most popular version being the strip-packing problem where rectangles are packed into a rectangular bin of *fixed* width but *infinite* height [2–4]. In our version of the problem, the user also specifies a *desired aspect ratio* for the resulting drawing so that the scaling that needs to be done before displaying the graph is minimal (Fig. 2).

One can find substantial literature on the design and analysis of algorithms for two-dimensional packing [2–5], the most popular version being *strip packing*. In strip packing, given list of $n \geqslant 1$ rectangles $L_n = (R_1, \ldots, R_n)$, each having dimensions (W_i, H_i), are to be packed into a semi-infinite strip of unit width. This problem has applications in many areas including stock-cutting, two-dimensional storage problems, and resource-constrained scheduling in computer systems [5]. Among the most popular are level algorithms such as *first-fit decreasing height* (FFDH) and *best-fit decreasing height* (BFDH).

One can find analysis of strip packing algorithms both in combinatorial form (i.e., worst-case analysis) and in probabilistic form (i.e., average case analysis) [2–5]. For an arbitrary list of n rectangles L_n, or simply L, all assumed to have width no greater than 1, let $\mathrm{OPT}(L)$ denote the minimum possible bin height within which rectangles in L can be packed. Also let $\mathrm{A}(L)$ denote the height actually used by a particular algorithm A when applied to L. The *wasted space* $\mathrm{WS}^A(L)$ is the unoccupied area of the packing: $\mathrm{WS}^A(L) = \mathrm{A}(L) - \sum_{i=1}^{n} W_i H_i$. Similarly, *fullness* of a packing $\mathrm{F}^A(L)$ expresses, in percentage, how effectively the area is used by the packing algorithm: $\mathrm{F}^A(L) = 100 \cdot (\sum_{i=1}^{n} W_i H_i)/\mathrm{A}(L)$. The aspect ratio of the packing produced by an algorithm A for rectangles L is denoted by $\mathrm{AR}^A(L)$. *Aspect ratio performance* of A, $\mathrm{ARP}^A(L)$, is defined as

$$\mathrm{ARP}^A(L) = \frac{\min(\mathrm{AR}^A(L), \mathrm{DAR}^A(L))}{\max(\mathrm{AR}^A(L), \mathrm{DAR}^A(L))},$$

where $\mathrm{DAR}(L)$ is the desired aspect ratio for packing of the rectangles L. The *adjust fullness* of a packing $\mathrm{AF}^A(L)$ ($\leqslant \mathrm{F}^A(L)$) expresses the fullness of a packing, in percentage, with respect to the desired aspect ratio. Fig. 3 illustrates this with an example, in which $\mathrm{F}^A(L) = A/(A+B)$, whereas $\mathrm{AF}^A(L) = A/(A+B+C)$.

When doing worst-case analysis, one is interested in finding the constants β and γ in the asymptotic performance bounds of the form: $\mathrm{A}(L) \leqslant \beta \cdot \mathrm{OPT}(L) + \gamma$ for all lists L. For average case analysis, on the other hand, one is interested in determining the function $f(n)$ for expressing the expected value of the height used by the algorithm: $E[\mathrm{A}(L)] = \sum_{i=1}^{n} W_i H_i + \Theta(f(n))$, where $\Theta(f(n))$ corresponds to $\mathrm{WS}^A(L)$.

For instance, for the level algorithm BFDH defined above, we have the following:

$$\mathrm{BFDH}(L) \leqslant 1.7 \cdot \mathrm{OPT}(L) + 1$$

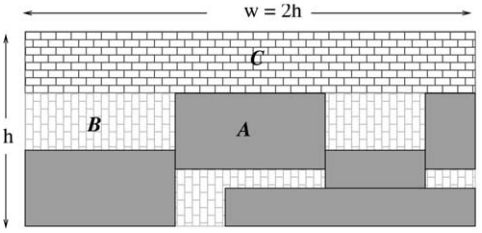

Fig. 3. Total area of the rectangles (A), area wasted in packing these rectangles (B), and additional area wasted when displayed in a region of aspect ratio 2 (C).

and

$$E[\mathrm{BFDH}(L)] = \frac{n}{4} + \Theta(\sqrt{n} \cdot \log^{3/4} n),$$

where all $2n$ variables $W_1, \ldots, W_n, H_1, \ldots, H_n$ are assumed to be independent, uniform random samples from the interval $[0, 1]$.

Two-way number partitioning problem is to divide a set of numbers into two subsets so that sum of the numbers in each subset are as nearly equal as possible. Efficient approximation algorithms for the problem are given in [6,7].

Given a set of nodes, *one-dimensional packing* along x-axis (y-axis) corresponds to the process of ordering these nodes with respect to their x-coordinates (y-coordinates) without any overlaps to *minimize* the total width (height) of the bounding rectangle. If the current relative positions of nodes are to be preserved in the packing, we call it *ordered one-dimensional packing*.

3. Strip-packing method

This method directly applies a known strip-packing algorithm such as BFDH. We calculate the width of the strip (equivalently, calculate the factor by which the rectangle dimensions are to be scaled) based on the desired aspect ratio, using the theoretical performance of the strip-packing algorithm. In other words, we calculate what the width of the strip (or the amount of scaling to be applied to the rectangles) should be such that the resulting packing yields an aspect ratio close to the desired one. When doing so, ideally, one would like to use the expected *average* performance of the strip-packing algorithm. However, the expected values for the average performance of these algorithms are of asymptotic form as opposed to an absolute one; so, we are forced to use *worst-case* bounds here.

For instance, for BFDH method described earlier, the absolute worst-case performance bound is $\mathrm{BFDH}(L) \leqslant 1.7 \cdot \mathrm{OPT}(L) + 1$. Assuming the worst case, one can easily scale the rectangle dimensions such that $1.7 \cdot \mathrm{OPT}(L') + 1 = 1/\mathrm{DAR}(L)$, where $L' = (R'_1, \ldots, R'_n)$ is the set of scaled rectangles with $R'_i = (W'_i, H'_i)$, $W'_i = s \cdot W$, $H'_i = s \cdot H$, $1 \leqslant i \leqslant n$, and $s \in \Re$ is the scaling factor. Notice that the desired aspect ratio will be achieved only when the worst-case performance is hit; otherwise, the aspect ratio will be larger than the desired. Fig. 4 illustrates how the size of the strip can be set or the dimensions of the rectangles can be scaled to obtain an aspect ratio closer to the desired one upon application of the BFDH algorithm with an example. Notice that both kinds of adjustments result in the same packing.

With this method, assuming object dimensions to be independent, uniform random samples from the interval $[0, 1]$ and $\mathrm{OPT}(L') \approx n/4$, the expected value of adjusted fullness can be calculated as follows:

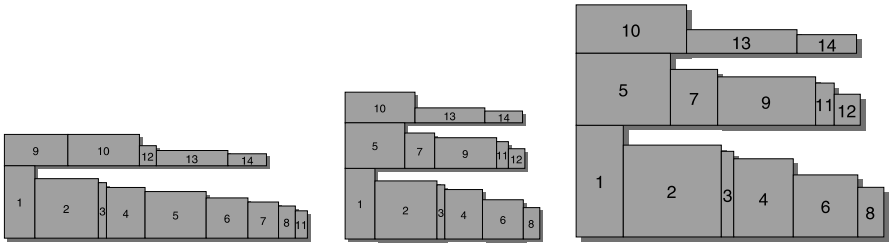

Fig. 4. Same set of rectangles strip-packed using BFDH with larger strip width (left), smaller strip width (middle), and strip width same as the one on left but with rectangles scaled to be larger.

$$E[\mathrm{AF}^{\mathrm{BFDH}}(L_n)] = 100 \cdot \frac{n}{4} \cdot \frac{1}{1.7 \cdot \frac{n}{4} + 1} \quad \Rightarrow \quad \lim_{n\to\infty} E[\mathrm{AF}^{\mathrm{BFDH}}(L_n)] = 58.8.$$

Similarly, the expected aspect ratio performance is calculated to be

$$E[\mathrm{ARP}^{\mathrm{BFDH}}(L_n)] = \frac{\min(E[\mathrm{AR}^{A}(L_n)], \mathrm{DAR}^{A}(L_n))}{\max(E[\mathrm{AR}^{A}(L_n)], \mathrm{DAR}^{A}(L_n))},$$

where

$$E[\mathrm{AR}^{A}(L_n)] = \frac{1}{E[\mathrm{BFDH}(L_n)]} = \frac{1}{\frac{n}{4} + \Theta(\sqrt{n} \cdot \log^{3/4} n)}$$

and

$$\mathrm{DAR}^{A}(L_n) = \frac{1}{1.7 \cdot \frac{n}{4} + 1}.$$

So,

$$\lim_{n\to\infty} E[\mathrm{ARP}^{\mathrm{BFDH}}(L_n)] = \frac{1}{4} \cdot \frac{4}{1.7} = 0.58.$$

Clearly, this algorithm is of $O(n \log n)$ time complexity. Fig. 5 shows experimental results, which verify the theoretical ones.

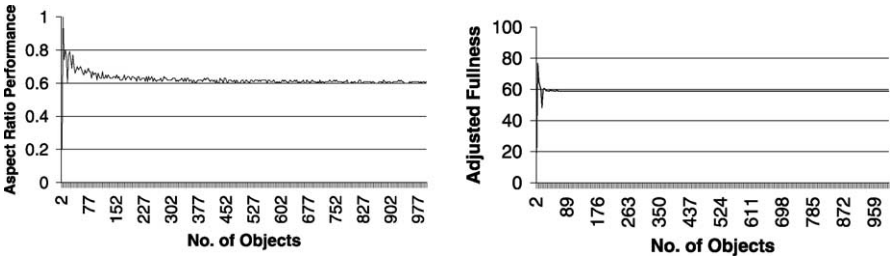

Fig. 5. Experimental results on strip-packing with BFDH.

4. Tiling: strip-packing with variable width strip

In this section we discuss a method which eliminates the need to "guess" the right size strip by maintaining a bin whose width dynamically changes. The algorithm starts by creating an initial level and placing the first rectangle in this level. It proceeds by determining whether the next rectangle in line should be added to one of the existing levels (the one which is the least utilized at the moment) or to a newly created level. The rectangle is tiled on an existing level if there is enough room. Otherwise, a decision is made on whether the current bin width should be enlarged or a new level should be formed, based on which choice results in an aspect ratio closer to the desired one. Fig. 6 shows an example application of the tiling algorithm. The rectangles are processed in the ascending order of their labels in this particular example.

In general, the tiling algorithm above does not assume any particular ordering of the objects. Our experiments show that when graph objects are sorted in nonincreasing height, most compact drawings are obtained (Fig. 7). Notice that when objects are processed in order of decreasing (i.e., nonincreasing) height, the algorithm turns into a variation of a strip-packing algorithm, BFDH to be more specific, where the strip width is dynamically increased as necessary to better fulfill the aspect ratio constraint.

Obviously, this algorithm is of $O(n \log n)$ time complexity.

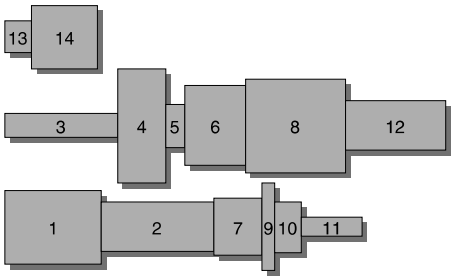

Fig. 6. An example of the application of the tiling algorithm.

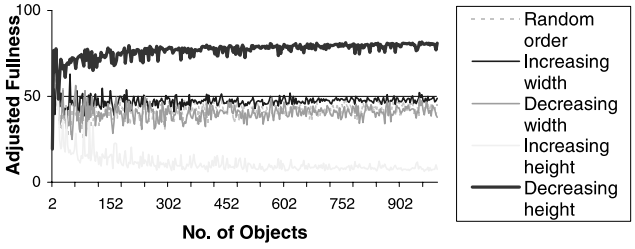

Fig. 7. How ordering objects by dimension affect the performance of tiling.

5. Alternate-bisection method

In this section we will investigate a method based on an alternate-bisection technique used in floorplanning in integrated circuit layout [8]. This divide-and-conquer method works by bisecting disconnected objects of a graph alternately as follows. The disconnected objects are bipartitioned and objects in each partition are recursively laid out. The recursion is to continue until a partition consists of a small, constant number of objects (e.g., one) whose optimal layout becomes easy if not trivial. At the end of each recursive step, when placing the two embedded partitions relatively, we alternate the orientation. For instance, the last step would place the two already positioned partitions side by side (horizontally) if the four partitions in the previous step were placed one on top of the other (vertically) pairwise.

Fig. 8 illustrates this method with an example. Even though this particular packing is quite balanced, there is a lot of room for improvement as the following analysis confirms. Let $T(n)$ and $W(n)$ denote total area required and area wasted by the algorithm described above for n objects, respectively. Let us take $n = 4k$, where k is an integer, for simplicity. Consider one step of the algorithm where four partitions already packed are put together (Fig. 8). Assuming object dimensions are independent, uniform random samples from the interval $[0, 1]$, we have $E[A_i] = 1/4$ for $i = 1, \ldots, 4$, and

$$E[A] = E[(\max(H_1, H_2) + \max(H_3, H_4)) \cdot (\max(W_1, W_3) + \max(W_2, W_4))],$$

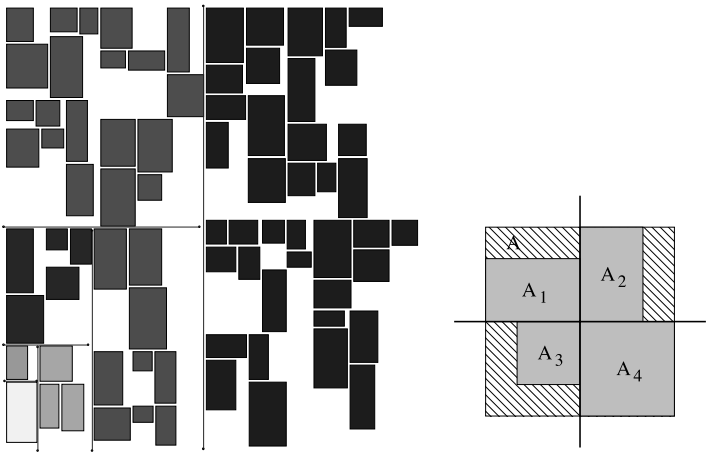

Fig. 8. On one thread of the recursion, alternately partitioned objects are shown with separating lines and colors (left). An illustration of how the four partitions of the objects recursively packed are put together (right).

which equals $(2/3 + 2/3)^2 = 16/9$ since a simple probabilistic analysis shows

$$\int\int H(x,y)f(x,y)\,\mathrm{d}x\,\mathrm{d}y = \int_0^1 \int_0^1 \max(x,y)\,\mathrm{d}x\,\mathrm{d}y$$
$$= \int_0^1 \left(\int_0^y y\,\mathrm{d}x + \int_y^1 x\,\mathrm{d}x \right) \mathrm{d}y = \frac{2}{3}$$

for two independent, uniform random variables (X, Y) with a joint pdf $f(x,y) = 1$ where $H(x,y) = \max(x,y)$. So, we have $T(4) = 16/9$ and $W(4) = 7/9$, and the recurrences: $T(n) = W(n) + (n/4)$ and $W(n) = 4\ W(n/4) + 4\ T(n/4)\ \frac{7}{9}$. Solving these recurrences, we get

$$W(n) = \mathrm{O}(n^{\log_4 64/9}) \approx \mathrm{O}(n^{1.41}).$$

Clearly, this is quite inefficient. However, when simple alternating ordered one-dimensional packings are applied in each recursive step, much more compact results are obtained as the experimental results show (Fig. 10). Fig. 9 shows the same set of objects packed without and with the use of ordered one-dimensional packings; they clearly make a positive difference.

In the above algorithm, partitioning of the objects could be done using one of the linear-time number partitioning heuristics mentioned earlier where each object corresponds to a number proportional to its size (e.g., area). Ordered one-dimensional packing, on the other hand, can be implemented in $\mathrm{O}(n \log n)$ time as described earlier. Overall, the time complexity of the algorithm will be $\mathrm{O}(n \log^2 n)$ since these two operations are performed for each bipartition of the original object list.

For independent, uniformly distributed random object dimensions, this algorithm will not favor one orientation over the other and yield "square-like" drawings. The desired aspect ratio can be respected by this algorithm by initially recursively partitioning the set of objects into two, one partition to be laid

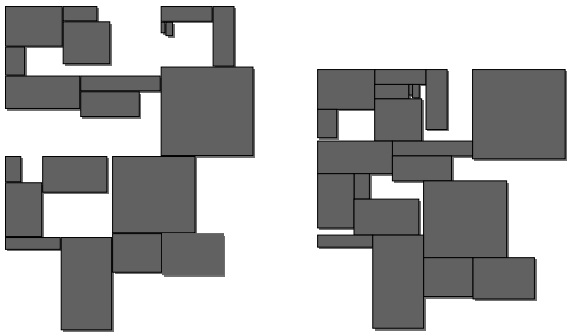

Fig. 9. The same set of objects packed with alternate bisection; ordered one-dimensional packings are not used in the left one.

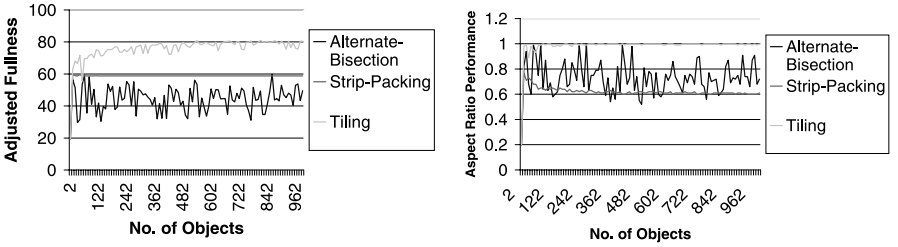

Fig. 10. Contrast of the performance of the three methods.

out with aspect ratio 1.0 and the other with $\mathrm{DAR}(L) - 1.0$ ($= (w - h)/h$), assuming $\mathrm{DAR}(L) = w/h > 1.0$.

We have experimented to find out how the performance of the algorithm depends on various size parameters that could be used during partitioning. Among these, area seems to yield best results.

6. Contrast of the methods

We have experimented with graphs laid out with random aspect ratio and random object dimensions, all independent and uniformly distributed. Fig. 10 shows the results obtained. During these experiments, settings that result in the best performance for each individual method were used. In general, the tiling method outperforms the other two not only for aspect ratio performance but also for adjusted fullness. The alternate-bisection method seems to yield better aspect ratio than the strip-packing method as expected; however, the strip-packing method clearly outperforms the alternate-bisection method in achieving better adjusted fullness in drawings.

Fig. 11 shows the same graph laid out with the three methods discussed. The desired aspect ratio is set to be 1.0. The tiling method seems to have performed best as the objects with this method appear to be largest when scaled to be displayed in a region with aspect ratio 1.0.

In the context of graph layout, one might argue that the object dimensions are not completely one of uniform distribution. Because the two types of disconnected objects, isolated nodes and components, in most cases will be of highly varying dimensions. We have also experimented with graphs where the nodes are of two different groups, where dimensions are uniformly distributed within each group but with different means (Fig. 12). Notice how the performance of the alternate-bisection method improved under such distribution.

In terms of execution time, both split-packing and tiling methods are superb since they are of $O(n \log n)$ complexity. The alternate-bisection method, on the

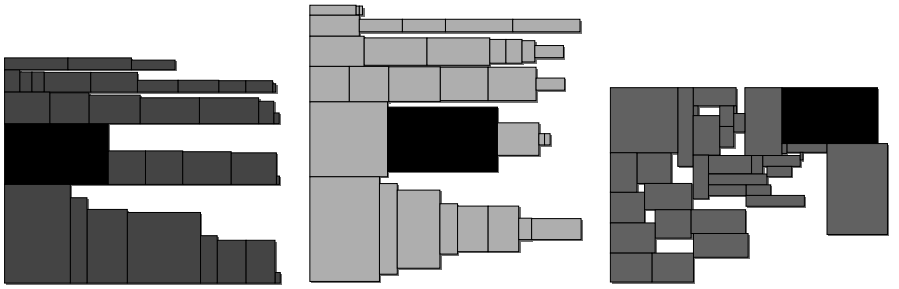

Fig. 11. The same graph laid out with split-packing, tiling, and alternate-bisection methods, respectively, all for desired aspect ratio 1.0.

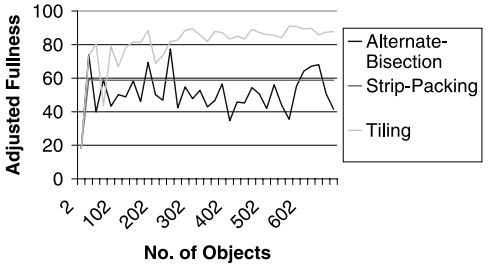

Fig. 12. Contrast of the performance in the context of graph layout.

other hand, gets a little slow as the number of objects are over a thousand. Considering most interactive graph drawing applications will not consist of more than, say one hundred, disconnected objects (isolated nodes and components), this method is also of practical value.

7. Conclusion

In this paper, we presented efficient algorithms for layout of disconnected objects in a graph for a specified aspect ratio. These near-linear algorithms are based on strip-packing, tiling, and alternate-bisection methodologies, and mostly make excellent use of the display area. The three methods perform in varying levels depending on certain parameters of the algorithms as well as the circumstances such as how the object dimensions are assumed to be distributed.

As future work, we would like to be able to take the specific shape of each object into account rather than simplifying all to be rectangles.

References

[1] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Algorithms for drawing graphs: an annotated bibliography, Comput. Geom. Theory Appl. 4 (1994) 235–282.
[2] E.G. Coffman, M.R. Garey, D.S. Johnson, R.E. Tarjan, Performance bounds for level-oriented two-dimensional packing algorithms, SIAM J. Comput. 9 (4) (1990) 808–826.
[3] B.S. Baker, E.G. Coffman, R.S. Rivest, Orthogonal packings in two dimensions, SIAM J. Comput. 9 (4) (1980) 846–855.
[4] E.G. Coffman, P.W. Shor, Packings in two dimensions: asymptotic average case analysis of algorithms, Algorithmica 9 (1993) 253–277.
[5] E.G. Coffman, M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing: an updated survey, in: G. Ausiello, M. Lucertini, P. Serafini (Eds.), Algorithm Design for Computer System Design, Springer, New York, 1984, pp. 49–106.
[6] R. Korf, From approximate to optimal solutions: a case study of number partitioning, in: in Proceedings of the 14th IJCAI, Montreal, Canada, 1995, pp. 266–272.
[7] G.S.L.N.K. Karmarker, R.M. Karp, A.M. Odlyzko, Probabilistic analysis of optimum partitioning, J. Appl. Probab. 23 (1986) 626–645.
[8] T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, Wiley, New York, 1990.